Sentiment-Driven Stock Predictor

Analysis of Stock Predictions Using Media Sentiment and Machine Learning on Financial Metrics

Venn Reddy
University of Georgia
Athens, GA, USA
vennreddy@uga.edu

Stephen Sulimani
University of Georgia
Athens, GA, USA
StephenSulimani@uga.edu

1 Abstract

Sentiment-Driven Stock Predictor (SDSP) is a sophisticated project that leverages sentiment analysis in conjunction with bootstrap aggregation techniques. The project is structured around two primary components. First, we have developed a class capable of accepting a stock's ticker symbol, along with specified start and end dates, to generate a comprehensive dataset. This dataset includes the stock's opening and closing prices, trading volume, various financial metrics, and, importantly, the daily average news sentiment associated with the stock.

2 Project Description

SDSP's process of creating a dataset is as follows:

1) Pull Stock Financial Data
2) Calculate Financial Metrics
3) Pull News Articles
4) Parse Each News Article
5) Quantify Sentiment for Each News Article
6) Add "Sentiment" Column to Stock DataFrame
7) Create Testing and Training Data for Stock
8) Train Bootstrap Aggregation Learner using 50 Random Or Decision Trees to Predict "Buy, Sell, or Hold" Signal, and Evaluate Accuracy
9) Output Model Performance Results to User

As we detail in the Challenges section, processing the large volume of data was a highly time-consuming task. For most of our tests, we retrieved approximately eleven months' worth of data for each stock, equating to around 230 trading days. For large-cap stocks like Apple (AAPL), this resulted in processing roughly 10,000 news articles.

While retrieving financial data and calculating financial metrics were relatively efficient tasks, parsing the news articles and determining their sentiment proved to be far more time-intensive. As later discussed, we sought to optimize these processes by reducing their time complexity. Additionally, we leveraged multiprocessing and threads to accelerate the execution of hundreds of these individual sub-tasks in parallel.

By optimizing these steps, we significantly reduced processing time. For instance, the time required to generate a dataset for Apple using eleven months of data was reduced from thirty-five minutes to sixteen minutes—a 54% improvement.

2.1 Research Process:

The initial phase of developing the Sentiment-Driven Stock Predictor (SDSP) involved extensive research. Our first challenge was identifying a reliable method for obtaining financial data, particularly as the widely used Python yfinance package is slated for deprecation. Given our limited budget of $0, we explored various alternatives and eventually discovered Stooq.com, a Polish-based platform offering free financial data across a broad range of securities, including stocks, bonds, and crypto-currencies.

The next hurdle was sourcing reliable news data for specific stocks on particular dates. Manually gathering articles from multiple sources would have been time-

consuming and inefficient. During our search for a suitable solution, we came across the Gödel Terminal. Marketed as an affordable alternative to the Bloomberg Terminal, the Gödel Terminal provided us with access to the news data we needed, allowing us to integrate it seamlessly into our project.

2.2 Development Phase and Implementation

The next stage in the development of SDSP focused on building the core functionality. Our primary objective was to abstract the entire process, ultimately aiming to create a single function and interface capable of generating an entire dataset with minimal input.

To begin, we developed a class to interface with Stooq.com. Since the platform did not offer a pre-existing API, we had to simulate standard user interactions on the website to retrieve the necessary data. Fortunately, acquiring a complete dataset in CSV format was as simple as making a single GET request to Stooq. While this function was straightforward to implement, we prioritized writing clean, maintainable, and easily understandable code to ensure long-term reliability and scalability.

Simultaneously, we sought a reliable method for fetching news data. We discovered that the Gödel Terminal provided access to a hidden backend API, which enabled us to retrieve news data for a specific stock ticker within a given date range. This discovery significantly streamlined the process of obtaining relevant articles for our project.

To facilitate interaction with this API, we developed a Godel class, similar to the one created for Stooq. Additionally, we created an Article class to efficiently process and manage the news data returned by the API. This design allowed us to retrieve news articles and represent them seamlessly as Python objects, enhancing our ability to manipulate and analyze the data. Throughout this phase, we continued to emphasize clean, modular, and well-documented code to ensure scalability and ease of maintenance.

Next, we focused on writing a series of methods to calculate specific financial metrics needed for training our models. We aimed to maintain consistency in function signatures so that we could easily pass in our Pandas DataFrame, have the functions calculate the relevant metric, and return the modified DataFrame.

To accomplish this, we created a Python module called financialcalc to store these functions. The first step was to define our target variable, Y, which represents the value we ultimately want our model to predict. We structured this as a classification problem, based on a daily return threshold. Our model will predict one of three actions: BUY, SELL, or HOLD. For example, if the threshold is set at 0.5%, the model will output BUY if it predicts the next day's closing price will exceed 0.5% above the current day's price. Conversely, if it expects the next day's closing price to be 0.5% lower, it will output SELL. If the model predicts little movement, it will output HOLD.

Within the financialcalc module, we implemented functions to calculate key financial metrics, including daily return and the daily Sharpe ratio.

The final step in building the SDSP prototype involved parsing the text from fetched Article objects and quantifying their sentiment. For this, we utilized the Natural Language Toolkit (nltk) Python package, along with the VADER (Valence Aware Dictionary and Sentiment Reasoner) lexicon, which allowed us to efficiently assess the sentiment of each article. After calculating the sentiment for all articles on a given day, we computed the mean sentiment and incorporated it into the stock’s Pandas DataFrame.

2.3 Challenges and Optimization

There were numerous challenges with our implementation. A major challenge throughout our project was runtime length: since we were querying all of the news articles for a stock over an entire year, we’d receive tens of thousands of results. Parsing these into Article objects and running sentiment analysis on each of these articles took a significant amount of time, so

we took multiple steps to try and increase the efficiency of these methods. For example, we modified our queryNews() method to return a dictionary of dates and respective Articles, instead of simply returning a list of Articles, in order to improve the efficiency of creating the "Signal" column in our create_sentiment_column() method. Here, we'll define some of our mathematical constraints to explain the boost in performance:

- n = Number of Trading Days Within Given Timeframe
- M = Total Number of Articles Pulled
- m = Avg number of Articles per date, where m = M/n

In our previous implementation, the create_sentiment_column() method iterated over all M articles for each of the n trading days to find articles matching each date, followed by evaluating the sentiments of all articles for that date. This approach had the following characteristics:

Outer Loop: n Iterations (Over Trading Days)
Inner Loop: M iterations (Over All Articles)
Time Complexity: O(n x M) = O(n x (n x m)) = **$O(n^2 x m)$**

In contrast, our new implementation still involves n iterations in the outer loop, but uses the dictionary to directly access the list of articles for each date, averaging m articles per date. This resulted in:

Outer Loop: n Iterations (Over Trading Days)
Inner Loop: m iterations (Over Articles for that Date)
Time Complexity: = **O(n x m)**

By restructuring our method to access articles by date directly, we reduced the time complexity from O(n x m^2) to O(n x m), a significant improvement. As mentioned earlier, in practice this led to a reduction in dataset construction time by about 50%, allowing users to create and test datasets more frequently.

3 Results

We tested our program on multiple stock tickers with the following parameters kept consistent:

Date Range: Jan 1, 2024 – Dec 1, 2024
Threshold: 0.8
Learners Used: Decision Tree Learners
Number of Learners Used: 25

Resulting output included the Accuracy, Precision, F1 Score, and Recall Score. For every day the models predicted "Buy" and the stock price went up the next day by > 0.8, or "Sell" and the stock price went down the next day by < 0.8, or "Hold" and the stock price was +-0.8 the next day, the model predicted correctly. Here are some of our results for different stock tickers:

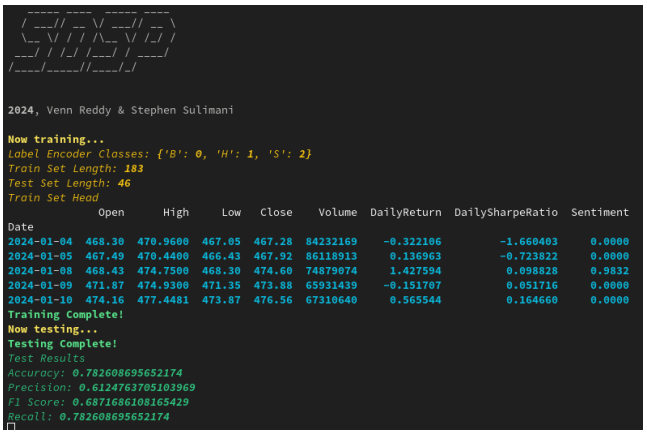


Figure 1: Prediction of Models on SPY

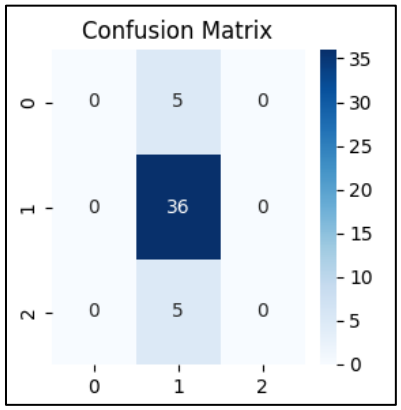
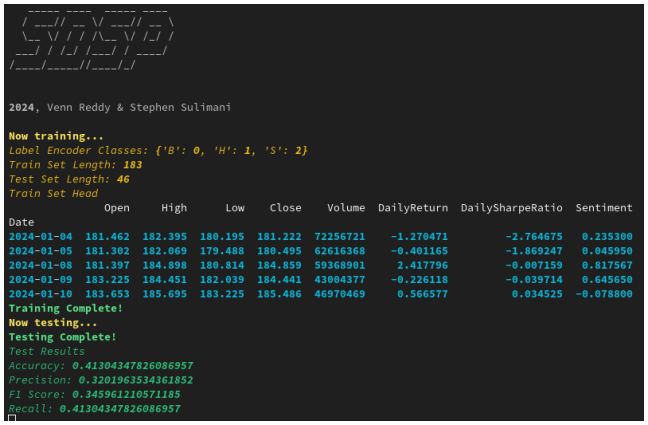


Figure 2: Confusion Matrix of SPY Predictions



```
2024, Venn Reddy & Stephen Sulimani

Now training...
Label Encoder Classes: {'B': 0, 'H': 1, 'S': 2}
Train Set Length: 183
Test Set Length: 46
Train Set Head
                Open    High     Low   Close    Volume  DailyReturn  DailySharpeRatio  Sentiment
Date
2024-01-04  181.462  182.395  180.195  181.222  72256721    -1.270471         -2.764675   0.235300
2024-01-05  181.302  182.069  179.488  180.495  62616368    -0.401165         -1.869247   0.045950
2024-01-08  181.397  184.898  180.814  184.859  59368901     2.417796         -0.007159   0.817567
2024-01-09  183.225  184.451  182.039  184.441  43004377    -0.226118         -0.039714   0.645650
2024-01-10  183.653  185.695  183.225  185.486  46970469     0.566577          0.034525  -0.078800
Training Complete!
Now testing...
Testing Complete!
Test Results
Accuracy: 0.41304347826086957
Precision: 0.3201963534361852
F1 Score: 0.345961210571185
Recall: 0.41304347826086957
```

Figure 3: Prediction of Models on AAPL

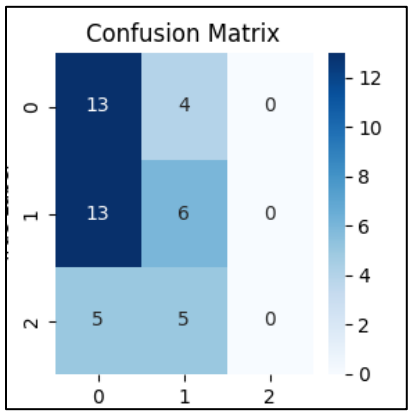
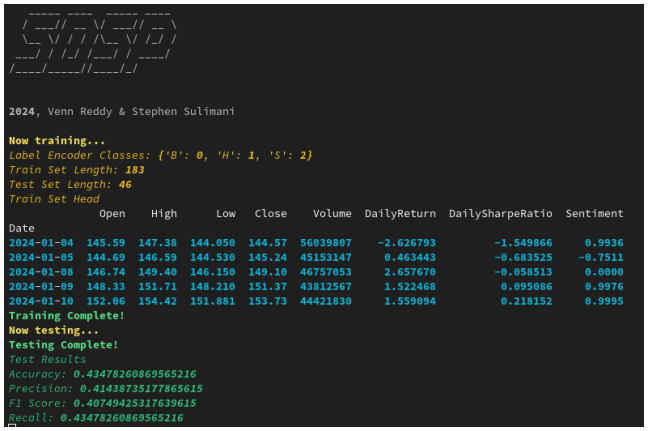


Figure 4: Confusion Matrix of AAPL Predictions



```
2024, Venn Reddy & Stephen Sulimani

Now training...
Label Encoder Classes: {'B': 0, 'H': 1, 'S': 2}
Train Set Length: 183
Test Set Length: 46
Train Set Head
                Open    High     Low   Close    Volume  DailyReturn  DailySharpeRatio  Sentiment
Date
2024-01-04  145.59  147.38  144.050  144.57  56039807    -2.626793         -1.549866     0.9936
2024-01-05  144.69  146.59  144.530  145.24  45153147     0.463443         -0.683525    -0.7511
2024-01-08  146.74  149.40  146.150  149.10  46757053     2.657670         -0.058513     0.0000
2024-01-09  148.33  151.71  148.210  151.37  43812567     1.522468          0.095086     0.9976
2024-01-10  152.06  154.42  151.881  153.73  44421830     1.559094          0.218152     0.9995
Training Complete!
Now testing...
Testing Complete!
Test Results
Accuracy: 0.43478260869565216
Precision: 0.41438735177865615
F1 Score: 0.40749425317639615
Recall: 0.43478260869565216
```

Figure 5: Prediction of Models on AMZN

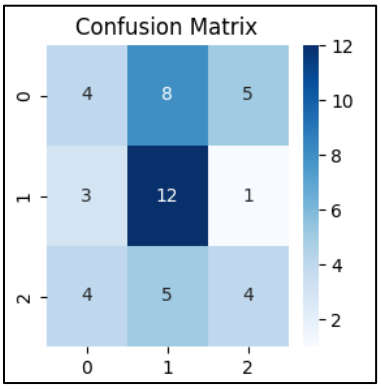
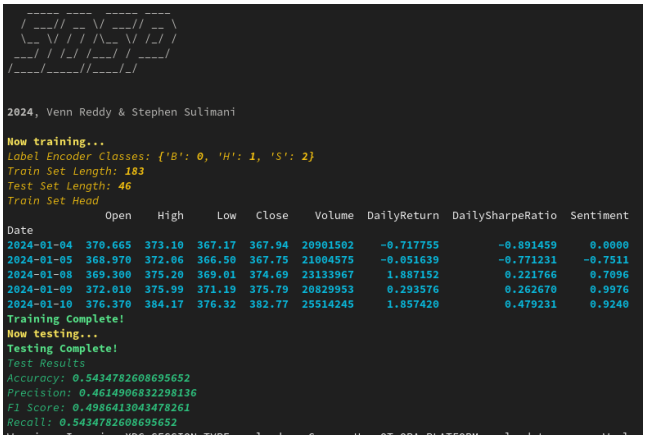


Figure 6: Confusion Matrix of AMZN Predictions



```
2024, Venn Reddy & Stephen Sulimani

Now training...
Label Encoder Classes: {'B': 0, 'H': 1, 'S': 2}
Train Set Length: 183
Test Set Length: 46
Train Set Head
                Open    High     Low   Close    Volume  DailyReturn  DailySharpeRatio  Sentiment
Date
2024-01-04  370.665  373.10  367.17  367.94  20901502    -0.717755         -0.891459     0.0000
2024-01-05  368.970  372.06  366.50  367.75  21004575    -0.051639         -0.771231    -0.7511
2024-01-08  369.300  375.20  369.01  374.69  23133967     1.887152          0.221766     0.7096
2024-01-09  372.010  375.99  371.19  375.79  20829953     0.293576          0.262670     0.9976
2024-01-10  376.370  384.17  376.32  382.77  25514245     1.857420          0.479231     0.9240
Training Complete!
Now testing...
Testing Complete!
Test Results
Accuracy: 0.5434782608695652
Precision: 0.4614906832298136
F1 Score: 0.4986413043478261
Recall: 0.5434782608695652
```

Figure 7: Prediction of Models on MSFT

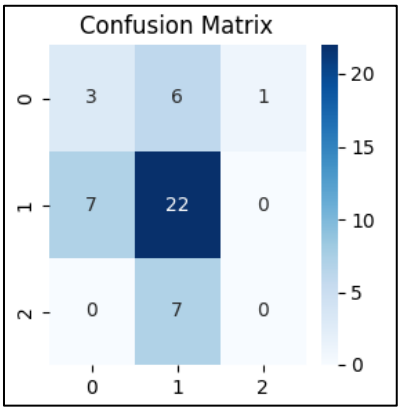


Figure 8: Confusion Matrix of MSFT Predictions

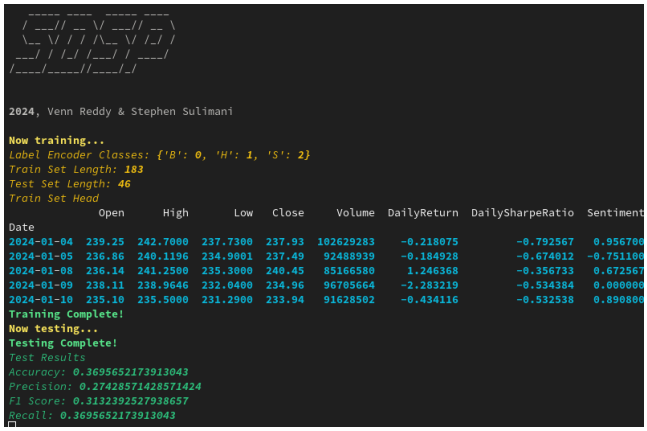


Figure 9: Prediction of Models on TSLA

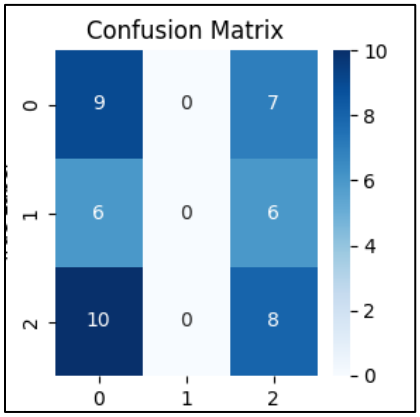


Figure 10: Confusion Matrix of TSLA Predictions

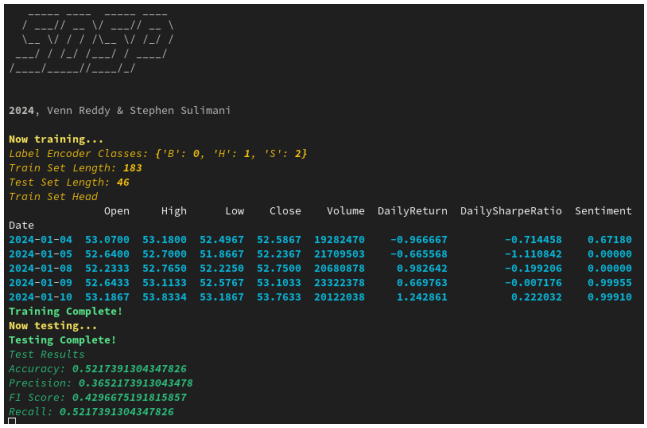


Figure 11: Prediction of Models on WMT

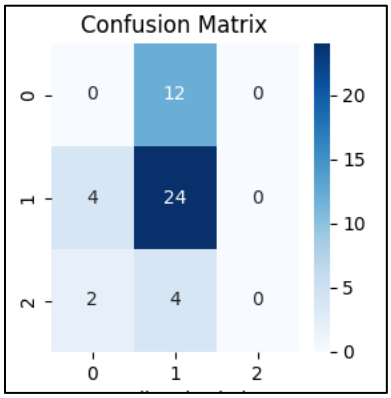


Figure 12: Confusion Matrix of WMT Predictions

4 Analysis

Through our experiments, we were able to identify a couple of key insights. First, was that as you increase the threshold, the models become more accurate. This makes intuitive sense—since the burden for deciding the purchase or sell a stock requires a higher price movement, it becomes more unlikely, and a "HOLD" signal becomes the most common prediction. If used as an investment tool, this would mimic a low-risk, low-reward investment strategy, since users would likely keep their investments for longer durations unless stocks went on significant runs.

As the threshold gets reduced, the model becomes less accurate, which is in-line with being a higher-risk, higher-reward investment strategy. Investors would receive more "BUY" and "SELL" signals, but it would become less accurate due to market volatility at those price ranges.

Another insight was related to the performance of the models on different types of stocks. On more volatile stocks, the model performed worse. For example, with TSLA, the model only achieved 36.9% accuracy, since it would frequently suggest a "HOLD" signal to investors, and the stock would fluctuate well above and below the threshold. On less volatile stocks, such as MSFT, WMT, and SPY, the model performed much better and consistently had accuracies ranging from 0.55~0.75. This was due to a somewhat high threshold value when testing, and the model predicting

"HOLD" more often for these stocks. This also intuitive—it's far less likely for Walmart or the SPY to move more than 0.8% percent in a day, whether it be because they are low-risk stocks or aggregate stocks.

5 Effort Justification

We believe this project meets the 1.5 times effort expectation per person because it combined financial calculations, bootstrap aggregation, sentiment analysis, and data management for pulling large amounts of news articles on financial data. By combining mathematical analysis with textual market analysis, we were able to simulate real-life processes of stock investors utilizing multiple sources of information to make informed decisions in our project, which went beyond the normal scope of mathematical analyses we performed in class.

5.1 Contributions

Venn: Responsible for sentiment package and sentiment analysis of news articles, building Signal DataFrame when passed articles, and created backup data import technique using yfinance (before we switched to using Stooq). Worked with Stephen on the Gödel package to parse Articles and construct dictionary of dates-to-articles in queryNews method with godel.py file. Also aided in researching the Godel Terminal and how to best retrieve information for use with later sentiment analysis. Helped create the UI for creating datasets and training models on them.

Stephen: Responsible for financial calc, builder, and learners packages. Worked to create python integrations for Stooq.com to pull financial data, and for Gödel Terminal to pull news data. Created builder class to abstract project and allow users to create datasets with one call per ticker, wrote functions to calculate financial metrics for predicting price movements and worked with Venn to create models to predict signals based off financial statistics and sentiment analysis data. Helped create the UI for creating datasets and training models on them.

Author's Note: We feel the work was split very evenly. We modularized the project structure into packages and divided the work between us, but also collaborated on multiple methods together because we needed our programs to interact correctly. Each member contributed equally to research as well, and we utilized version control software (Git) to complete collaborative work remotely.

6 Checklists of Rubric Requirements

6.1 Checklist – Stephen Sulimani

1) **Code Complexity and Creativity**: Thought of a creative solution to simulate real-world user interaction to fetch data from Stooq (who did not have an API) and Godel (who had a hidden API)
2) **Ability to Complete Milestone Tasks**: Completed all expected work before Milestone tasks and deadlines
3) **Real-World Application of Project**: While results for using SDSP have not "solved" trading, the ability to create a dataset with these features in a quick and simple way is very useful for machine learning and investment knowledge
4) **Presentation of Data**: Worked with Venn to present data through a clean UI that allows users to easily make, store, access, and train models on datasets for specific stocks at specified date ranges.
5) **Problem Solving Approach**: Worked with Venn to implement numerous standards and good coding practices. Includes abstraction of code, usage of modules, optimization of methods, clear and accessible documentation, and remote programming via Git.

6.1 Checklist – Venn Reddy

1) **Code Complexity and Creativity**: Thought of a creative solution to optimize sentiment analysis and DataFrame construction when dealing with large amounts of Articles.
2) **Ability to Complete Milestone Tasks**: Completed all expected work before Milestone tasks and deadlines
3) **Real-World Application of Project**: Mimics the process of using financial metrics and contextual metrics (such as media favorability of a stock and public perception) to make more informed investments and provides DataFrames with useful information.

4) **Presentation of Data**: Worked with Stephen to present data through a clean UI that allows users to easily make, store, access, and train models on datasets for specific stocks at specified date ranges.
5) **Problem Solving Approach**: Worked with Stephen to implement numerous standards and good coding practices. Includes abstraction of code, usage of modules, optimization of methods, clear and accessible documentation, and remote programming via Git.